图像处理

（直方图均衡化与锐化处理）

电院2372 刘浩宇

# 背景

由于图像获取, 传输过程中信号的损失和噪声的引入, 图像很容易获得不好的效果。我们需要使用一些算法进行图像增强 (Image enhancement) , 这样可以帮助我们获得更好的视觉效果。直方图均衡化主要调整画面亮度的分布，图像锐化则通过一些卷积的做法增强图像边缘。

# 二．直方图均衡化

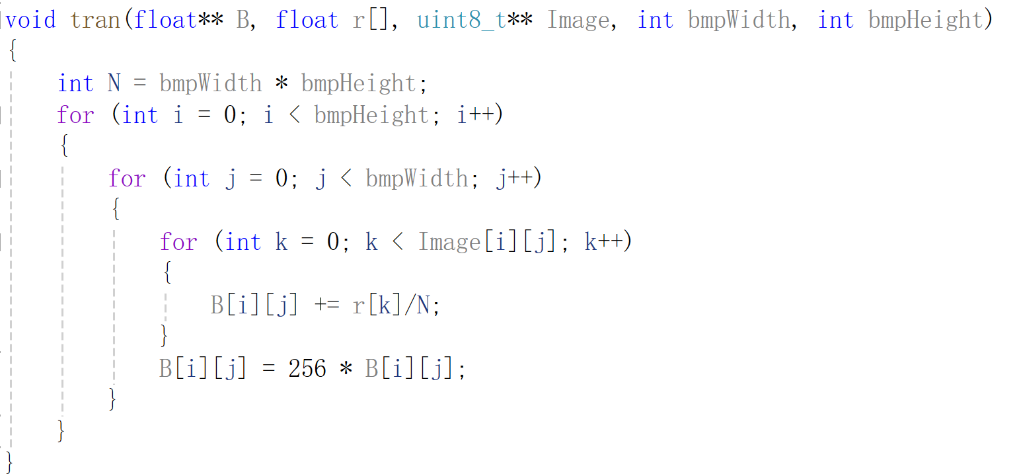
1.代码实现

整个代码分为四部分：

1. main函数：定义图像的宽，高，三维，二维像素值数组。进行函数的调用，动态数组的定义与删除。
2. trangray头文件：包含四个函数，分别实现：RGB图像转化为灰度图，收集均衡后的像素值数组，对像素值进行均衡化，灰度图转化为RGB图像。
3. intput\_output头文件：包含两个函数，进行图像的读取和保存。
4. getr头文件：包含两个函数，分别实现：统计各像灰度包含像素个数，画出灰度直方图（用到opencv库）。

2.主体算法

本算法的主体为对图像各个像素点灰度值的均衡化，在实现过程中，我选择利用二维动态浮点数组储存原图像各像素灰度值并对这个数组进行处理：



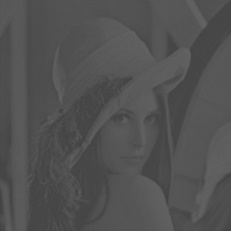
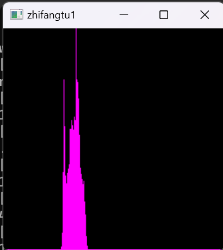
其中r[256]中储存了每一灰度值包含的像素点个数，N为图像像素点总个数。

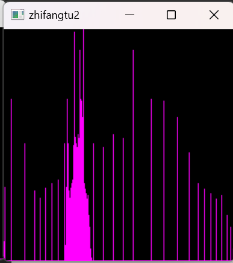
均衡后的B[ i ][ j ]=256\*Σk（0~D[ i ][ j ]）r[ k ]。（D[ i ][ j ]为第i行第j列像素的灰度值）

之后再将二维数组B的值传递向uint8\_t的二维数组，进而通过tranRGB函数转化为RGB图像保存。

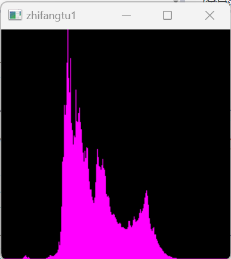
3.算法效果

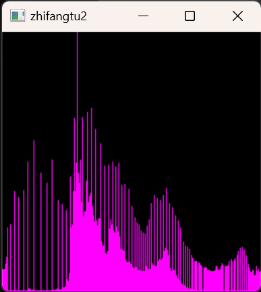
我们以两张图片为例：

* 1. bmp 其原灰度直方图：

经过该算法处理后的灰度直方图：

处理后的图像：

（2）11.bmp其原灰度直方图：

经过该算法处理后的灰度直方图：

处理后的图像：

从这两个例子可以看出，经过这一算法处理后的图片对比度和清晰度都有了较明显的提升，且原来较模糊的图片改进更大。从灰度直方图来看，处理后的图像灰度值在保留原来趋势的情况下分布更加均匀合理，因此图像看起来更加清晰。

4.算法实现过程中的问题及解决

（1）直方图的绘制

最初我考虑的是直接输出各灰度值包含的像素点个数，发现像素点个数庞大且256个灰度值及其对应的个数输出后冗长且不直观，就去网上寻求画图方法。查询相关资料后，选择了下载并配置opencv库并用其画出直观的图像。

（2）均衡化算法

根据老师提供的PPT的算法指导，我选择将每一灰度值对应的像素数目进行归一化（即上方提到的r[ k ]/N）后乘以256，从而确保像素灰度值落在0~255之间且实现均衡化。

1. **图像锐化**
2. 代码实现

整个代码分为五部分：

1. main函数：定义图像的宽，高，三维，二维像素值数组。进行函数的调用，动态数组的定义与删除。
2. intput\_output头文件：包含两个函数，进行图像的读取和保存。
3. getI头文件：包含两个函数，进行像素值增广矩阵的获得和锐化后图像像素矩阵的获取。
4. tran头文件：包含两个函数，实现灰度图与RGB图像之间的相互转化。
5. juanji头文件：包含三个全局数组和三个函数，分别实现三种卷积（拉普拉斯卷积，平滑卷积，检测竖线卷积）
6. 主体算法

本算法的主体为图像各像素点灰度值的卷积化，在算法中提供了三种不同的卷积方式：拉普拉斯卷积，平滑卷积和检测竖线卷积。分别用到三个卷积核以及相关实现代码均在juanji头文件中。代码较多，这里放第一种卷积的算法，另外两种同理，只是使用的卷积核不同

float ker0[3][3] = { 0,-1,0,-1,5,-1,0,-1,0 };

void juanji0(float\*\* T, float\*\* I, int width, int height)

{

for (int i = 1; i < height; i++)

{

for (int j = 1; j < width; j++)

{

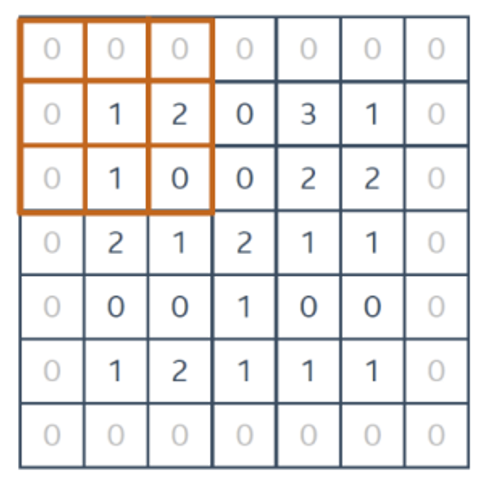
T[i - 1][j - 1] = I[i - 1][j - 1] \* ker0[0][0] + I[i][j - 1] \* ker0[0][1] + I[i + 1][j - 1] \* ker0[0][2] + I[i - 1][j] \* ker0[1][0] + I[i][j] \* ker0[1][1] + I[i + 1][j] \* ker0[1][2] + I[i - 1][j + 1] \* ker0[2][0] + I[i][j + 1] \* ker0[2][1] + I[i + 1][j + 1] \* ker0[2][2];

}

}

}

其基本原理就是对于原图像各像素点的灰度值构成的矩阵进行增广化，使其变为外围均为0的一个浮点矩阵：

之后再用另一个浮点矩阵储存卷积处理后的灰度值。

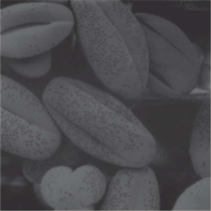
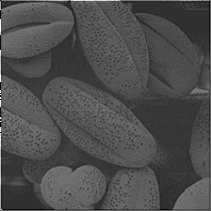
另外两个卷积核：

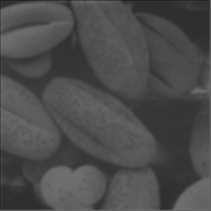
平滑卷积： ker1[3][3]={ {0.1111,0.1111,0.1111},{0.1111,0.1111,0.1111},{0.1111,0.1111,0.1111} };

检测竖线：ker2[3][3] = { {1,0,-1},{1,0,-1}, {1,0,-1} };

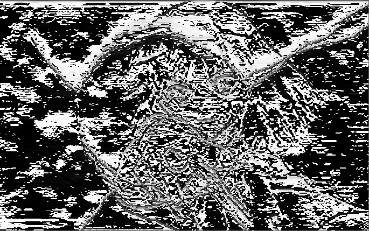
1. 算法效果

仍然以两张图片为例，展示三种锐化方式的效果：

* 1. bmp 拉普拉斯卷积：

平滑卷积： 检测竖线：

（2）11.bmp 拉普拉斯卷积：

平滑卷积： 检测竖线：

两组图片的示例展现出三种算法的不同之处：

拉普拉斯卷积旨在适当突出图像中的线条和细节，使图像更加明晰；平滑卷积削弱了图像中线条的锋锐，使图像更加平滑；而检测竖线突出图像中的每一根线条但是忽视图案的清楚度。

三种算法各自有不同的应用场景，需要根据实际情况来使用。

1. 算法实现过程中遇到的问题

在该算法实现的过程中唯一的问题就是数组下标的限制，因为卷积算法中对于某一像素点来说，周围八个像素都要参与运算，这对于数组边界的控制有着更高的要求，我在处理过程中选择另外创建一个外围一圈元素均为0的增广数组来解决数组边界元素值的计算。